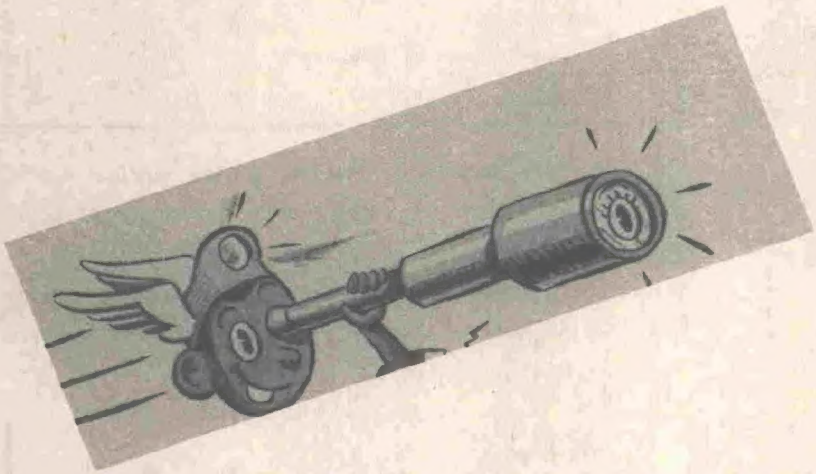
Unix

ไพโรจน์ ไววานิชกิจ

Shell Programming กับ UNIX ตอนที่ 2



เรื่องราวและประเภทของตัวแปรที่มีใช้งานในระบบปฏิบัติการ UNIX อ้างอิงถึงเชลล์แบบบอร์น

จากบทความในตอนที่ 1 ผู้เขียนได้กล่าวถึงแนวคิดของ Shell Programming และเรื่องราวของโปรเซส เพื่อเป็นพื้นฐานเบื้องต้นสำหรับผู้อ่านไปแล้ว ในฉบับนี้จะเป็นเรื่องของตัวแปร (variable) ซึ่งถือเป็นสิ่งสำคัญประการหนึ่งในการเขียนโปรแกรม ผู้ใช้งานระบบปฏิบัติการ UNIX สามารถใช้ประโยชน์จากตัวแปรในการส่งผ่านข้อมูล หรือนำตัวแปรไปใช้งานกับคำสั่งเงื่อนไขต่าง ๆ เพื่อให้ผลลัพธ์ที่ได้จากการทำงานของโปรแกรมเป็นไปตามที่ต้องการ ในขั้นต้นนี้ผู้เขียนจะขอกล่าวถึงเรื่องราวและประเภทของตัวแปรที่มีใช้งานในระบบปฏิบัติการ UNIX อนึ่ง ตัวอย่างต่าง ๆ ที่มีการกล่าวถึงในบทความตอนนี้จะอ้างอิงถึงเชลล์แบบบอร์น (Bourne Shell) เป็นหลัก

ในเรื่องของระบบปฏิบัติการและการเขียนโปรแกรมในระบบเชลล์ใด ๆ ก็ตาม ผู้ใช้งานมักหลีกไม่พ้นที่จะต้องเข้าไปเกี่ยวข้องกับการใช้งานตัวแปร โดยทั่วไปผู้ใช้งานมักทำการสร้างและกำหนดค่าให้กับตัวแปรเพื่อนำไปใช้งานและอ้างอิงถึงในการทำงานหรือในโปรแกรมที่ได้รับการเขียนขึ้น ระบบปฏิบัติการ UNIX ได้กำหนดประเภทของตัวแปรไว้ทั้งสิ้น 3 ชนิดโดยมีรูปแบบการใช้งานที่แตกต่างกัน ตัวแปรชนิดแรกคือตัวแปรชนิดผู้ใช้งานเป็นผู้กำหนดเอง (User-Created Variable) เป็นตัวแปรที่ผู้ใช้งานสามารถกำหนดชื่อและตั้งค่าได้ตามต้องการโดยอิสระ ทั้งนี้ผู้ใช้งานยังสามารถทำการเปลี่ยนแปลงค่าของตัวแปรนี้ได้ทุกเมื่อและยังสามารถกำหนดให้ตัวแปรเป็นแบบอ่านค่าได้อย่างเดียวเพื่อป้องกันการถูกเปลี่ยนแปลงค่าในอนาคต ตัวแปรประเภทต่อมาคือตัวแปรมาตรฐาน (Keyword Variable) ซึ่งเป็นตัวแปรที่ถูกสร้างโดยระบบปฏิบัติการ UNIX ตัวแปรประเภทนี้จะมีชื่อเฉพาะและถูกใช้งานโดยระบบปฏิบัติการ โดยเมื่อผู้ใช้งานทำการ login เข้าสู่ระบบปฏิบัติการ เชลล์ที่ทำหน้าที่ควบคุมการใช้งานจะกำหนดค่าให้กับตัวแปรมาตรฐานให้กับผู้ใช้งานทันที ตัวอย่างของตัวแปรในกลุ่มนี้ก็คือ **HOME** ซึ่งเก็บค่าไดเรคทอรีประจำตัวผู้ใช้งาน **PATH** เก็บรายชื่อของไดเรคทอรีที่เชลล์จะทำการค้นหาชื่อของคำสั่งที่ผู้ใช้งานป้อนผ่านคอนโซล แม้ว่าตัวแปรเหล่านี้จะได้รับการกำหนดค่าเริ่มต้นจากเชลล์ก็ตาม ผู้ใช้งานก็ยังมีสิทธิทำการปรับเปลี่ยนค่าให้เป็นไปตามต้องการได้ทุกเมื่อ วิธีหลัก ๆ ที่ใช้ก็คือการแก้ไขค่าของตัวแปรมาตรฐานเหล่านี้ซึ่งได้รับการกำหนดค่าในไฟล์ .profile ซึ่งอยู่ในไดเรคทอรีใช้งานหลักของผู้ใช้งานแต่ละราย สำหรับตัวแปรประเภทสุดท้ายก็คือตัวแปรเชลล์ที่อ่านค่าได้เพียงอย่างเดียว (Readonly Shell Variable) ผู้ใช้งานสามารถกำหนดชื่อของตัวแปรเหล่านี้ได้โดยการใช้อักขระพิเศษต่าง ๆ เช่น $? หรือ $# โดยทั่วไปจะใช้ตัวแปรเหล่านี้ในการรับค่าของอาร์กิวเมนต์ซึ่งตามหลังการใช้งานคำสั่งต่าง ๆ ผู้ใช้งานไม่สามารถทำการกำหนดค่าให้กับตัวแปรกลุ่มนี้ได้โดยตรงเหมือนดังเช่นในกรณีของตัวแปรประเภทอื่น ๆ

เมื่อได้รับทราบถึงการจำแนกประเภทของตัวแปรออกเป็นกลุ่ม ๆ แล้ว ผู้เขียนจะขอกล่าวถึงการใช้งานตัวแปรทั้ง 3 ประเภทเรียงตามลำดับไปดังนี้

ตัวแปรชนิดผู้ใช้งานสร้างเอง

ผู้ใช้งานสามารถกำหนดชื่อของตัวแปรประเภทนี้ขึ้นได้จากการนำตัวอักษรหรือตัวเลขมาประกอบกัน โดยมีข้อแม้ว่าชื่อของตัวแปรเหล่านี้จะต้องเริ่มต้นด้วยตัวอักษรเท่านั้น ผู้เขียนขอยกตัวอย่างการกำหนดค่าตัวแปรประเภทนี้ในกรอบตัวอย่างที่ 1 โดยจะกำหนดตัวแปรชื่อ person และกำหนดค่าให้เป็น wallaya ข้อควรจำก็คือผู้ใช้งานจะต้องไม่มีการกดปุ่ม SPACE หรือ TAB นำหน้าหรือตามหลังเครื่องหมาย "=" ผู้เขียนจะแสดงยืนยันค่าของตัวแปร person โดยการใช้คำสั่ง echo ขอให้สังเกตความแตกต่างในการแสดงผลระหว่างการใช้คำสั่ง echo ตามด้วย person เทียบกับการใช้คำสั่ง echo ตามด้วย $person

กรอบที่ 1

```
$ person=wallaya
$ echo person                ← ให้พิมพ์ข้อความว่า person ออกบนหน้าจอ
person
$ echo $person               ← ให้ค่าของตัวแปร person ออกบนหน้าจอ
wallaya
```

จากตัวอย่างในคำสั่งที่สองและคำสั่งที่สามแสดงให้เห็นว่าในการที่ต้องการจะอ้างถึงค่าของตัวแปรนั้น ผู้ใช้งานจะต้องนำหน้าชื่อของตัวแปรด้วยเครื่องหมาย $ ทั้งนี้เมื่อใช้คำสั่ง echo $person เชลล์จะรับทราบว่า person เป็นตัวแปรที่ได้รับการกำหนดค่าขึ้น มิได้เป็นเพียงข้อความธรรมดาที่จะถูกนำไปแสดงค่าบนหน้าจอ เชลล์จะทำการแทน (substitute) คำว่า person ด้วยค่าของตัวแปรดังกล่าวก่อนที่จะถูกนำส่งไปปรากฏบนหน้าจอ ในกรณีที่ผู้ใช้งานต้องการอ้างถึงชื่อของตัวแปรแทนที่จะให้เป็นค่าของตัวแปรก็สามารถทำได้โดยการครอบ $person ด้วยเครื่องหมายคำพูดเดี่ยว (' ') หรือเครื่องหมาย \ การใช้เครื่องหมายคำพูดคู่ ("") จะให้ผลเช่นเดียวกับการไม่ใช้เครื่องหมายใด ๆ ทั้งนี้ขอให้ดูตัวอย่างซึ่งแสดงในกรอบตัวอย่างที่ 2

กรอบที่ 2

```
$ echo $person
wallaya
$ echo "$person"
wallaya
$ echo '$person'
$person
$ echo \$person
$person
```

ประโยชน์ของเครื่องหมายคำพูดคู่ที่มีการนำมาใช้งานอย่างมากก็คือการยกเลิกการแทนความหมายของเครื่องหมายพิเศษอื่น ๆ ลองพิจารณาการกำหนดค่าให้กับตัวแปร person ในกรอบตัวอย่างที่ 3 เมื่อใช้คำสั่ง echo เพื่อแสดงค่าของตัวแปร $person โดยครอบค่าตัวแปรดังกล่าวด้วยเครื่องหมายคำพูดคู่จะส่งผลให้เชลล์ไม่ทำการแสดงเครื่องหมายพิเศษที่อยู่ในตัวแปร person ซึ่งก็คือเครื่องหมายคำพูดคู่ที่มีการกำหนดค่าไว้ตั้งแต่แรกนั่นเอง

กรอบที่ 3

```
$ person="wallaya and nongnuch"
$ echo "$person"
wallaya and nongnuch
```

ในกรณีที่ค่าของตัวแปรที่มีการเว้นวรรคด้วยเครื่องหมาย SPACE หรือ TAB ที่อยู่ติดกันมากกว่าหนึ่งตัว การอ้างถึงค่าของตัวแปรในรูปของ $person และ "$person" จะให้ผลแตกต่างกัน โดยการอ้างถึงโดยไม่มีเครื่องหมายคำพูดคู่จะเป็นการสั่งให้เชลล์แทนค่าเครื่องหมาย SPACE หรือ TAB ที่อยู่ติดกันมากกว่าหนึ่งนั้นด้วยเครื่องหมาย SPACE เพียงตัวเดียวดังแสดงในกรอบตัวอย่างที่ 4

กรอบที่ 4

```
$ person="wallaya    and                nucharee"
$ echo "$person"
wallaya    and          nucharee
$ echo "$person"
wallaya and nucharee
```

เชลล์จะไม่สามารถตีความหมายของเครื่องหมายพิเศษบางอย่าง เช่น * และ ? เมื่อมีการระบุอยู่เป็นส่วนหนึ่งของค่าตัวแปรได้ แต่เมื่อมีการอ้างถึงค่าของตัวแปร การตีความหมายก็จะเป็นไปได้ตามปกติ ขอให้ดูจากกรอบตัวอย่างที่ 5 จะเห็นว่าเมื่อมีการอ้างถึงตัวแปร $memo โดยไม่ใช้เครื่องหมายคำพูดคู่ เชลล์จะทำการแทนเครื่องหมาย * ด้วยการจับคู่กับไฟล์ที่มีอยู่ในไดเรคทอรีใช้งานปัจจุบันซึ่งเริ่มต้นชื่อด้วยคำว่า pairoj อันได้แก่ไฟล์ pairoj.report, pairoj.summary และ pairoj.letter และเมื่อใดก็ตามที่มีการอ้างถึงตัวแปรดังกล่าวโดยครอบด้วยเครื่องหมายคำพูดคู่ เชลล์จะทำการแสดงผลด้วยค่า pairoj* การครอบตัวแปรด้วยเครื่องหมายคำพูดนี้มีศัพท์เรียกเป็นทางการว่าการโหวด (Quoted)

กรอบที่ 5

```
$ memo=pairoj*
$ echo "$memo"
pairoj*
$ echo $memo
pairoj.report   pairoj.summary          pairoj.letter
$ ls
pairoj.report   pairoj.summary          pairoj.letter
```

เมื่อได้รับทราบถึงวิธีการสร้าง การกำหนดค่า และการนำตัวแปรมาใช้งานแล้ว ในกรณีที่ผู้ใช้งานมีความคิดว่าตนไม่ต้องการจะใช้งานตัวแปรบางตัวและต้องการยกเลิกตัวแปรดังกล่าวนั้นออกจากระบบปฏิบัติการก็สามารถทำได้ 2 วิธี วิธีแรกคือการกำหนดค่าให้กับตัวแปรโดยไม่ต้องพิมพ์ข้อความหรือค่าใด ๆ ตามหลังเครื่องหมาย "=" สำหรับวิธีที่สองก็คือการใช้คำสั่ง **unset** ตามด้วยชื่อของตัวแปรที่ต้องการยกเลิก ทั้งนี้ขอให้ดูตัวอย่างการใช้งานจากกรอบตัวอย่างที่ 6

กรอบที่ 6

```
$ person=
$ echo $person
$
$ unset memo
$ echo $memo
$
```

มาถึงจุดนี้ผู้เขียนก็ใคร่จะขอแนะนำคำสั่งพิเศษที่ได้รับการกำหนดขึ้นเพื่อให้ใช้งานกับตัวแปรทุกประเภทซึ่งมีอยู่ทั้งสิ้น 3 คำสั่ง คำสั่งทั้งสามนี้เป็นคำสั่งที่มีติดตัวมากับเชลล์แบบบอร์น โดยแบ่งออกเป็นคำสั่ง **readonly** คำสั่ง **export** และคำสั่ง **read** อันมีรายละเอียดดังต่อไปนี้

คำสั่ง readonly เป็นคำสั่งที่ใช้กำหนดให้ตัวแปรมีคุณสมบัติเป็นแบบอ่านค่าได้เพียงอย่างเดียว ไม่สามารถแก้ไขค่าได้ ตัวอย่างในกรอบที่ 7 จะแสดงให้เห็นถึงการเปลี่ยนคุณสมบัติของตัวแปร person ทั้งนี้ผู้ใช้งานจะต้องทำการสร้างและกำหนดค่าให้กับตัวแปรก่อนที่จะใช้คำสั่ง **readonly** และเมื่อมีการเปลี่ยนแปลงคุณสมบัติไปแล้ว ผู้ใช้งานจะไม่สามารถแก้ไขหรือเปลี่ยนแปลงค่าของตัวแปรดังกล่าวได้อีก หากมีความพยายามจะทำการเปลี่ยนแปลง เชลล์จะทำการส่งข้อความเตือนกลับมา การใช้คำสั่ง **readonly** โดยไม่ตามด้วยชื่อตัวแปรใด ๆ จะทำให้เชลล์แสดงรายชื่อของตัวแปรทั้งหมดที่ได้รับการกำหนดคุณสมบัติให้เป็นแบบอ่านค่าได้เพียงอย่างเดียว ในกรณีที่ผู้ใช้งานกำหนดคุณสมบัติให้ตัวแปรมาตรฐานเป็นแบบอ่านค่าได้เพียงอย่างเดียว เชลล์ก็จะแสดงชื่อของตัวแปรมาตรฐานดังกล่าวนั้นขึ้นมาด้วย อนึ่ง คำสั่ง readonly จะไม่แสดงรายชื่อของตัวแปรเชลล์ที่อ่านค่าได้เพียงอย่างเดียว (เช่น $?, $#)

กรอบที่ 7

```
$ person=kong
$ echo $person
kong
$ readonly person
$ person=bundhit
person: is read only
```

คำสั่ง export เนื่องจากค่าของตัวแปรที่ได้รับการกำหนดขึ้นในโปรเซสแต่ละชุดนั้นต่างก็มีความหมายเฉพาะในกลุ่มโปรเซสนั้น และเนื่องจากเชลล์สคริปต์ที่เป็นโปรเซสลูกจะไม่สามารถรับทราบถึงค่าของตัวแปรต่าง ๆ ที่ผู้ใช้งานได้กำหนดขึ้นหลังจากที่ได้ทำการ login จนกว่าผู้ใช้งานจะได้ทำการส่งผ่านตัวแปรเหล่านั้น วิธีที่ใช้ในการส่งผ่านตัวแปรให้โปรเซสลูกที่จะเกิดขึ้นต่อไปก็คือการใช้คำสั่ง **export** ตามด้วยชื่อของตัวแปรที่ต้องการ ซึ่งจะทำให้เชลล์นำค่าของตัวแปรที่ได้รับการส่งผ่านนี้ไปเก็บไว้ในสภาวะแวดล้อม (environment) ของโปรเซสลูกที่เกิดตามมา ซึ่งเป็นการทำให้โปรเซสลูกทั้งหมดสามารถรับรู้ได้ถึงการอ้างค่าตัวแปรเหล่านี้ ขอให้ดูตัวอย่างการอ้างค่าตัวแปรระหว่างสคริปต์ extest1 ซึ่งถือเป็นโปรเซสแม่กับสคริปต์ subtest ซึ่งเป็นโปรเซสลูกภายในกรอบตัวอย่างที่ 8

กรอบที่ 8

```
$ cat extest1
cheese=american
echo "extest1 1: $cheese"
subtest
echo "extest1 2: $cheese"
$ cat subtest
echo "subtest 1: $cheese"
cheese=swiss
echo "subtest 2: $cheese"
$ extest1
extest1 1: american
subtest 1:
subtest 2: swiss
extest1 2: american
```

จากตัวอย่างข้างต้นจะเห็นว่าสคริปต์ **subtest** ไม่เคยได้รับค่าของตัวแปร cheese มาจากสคริปต์ **extest1** ในขณะที่ **extest1** ก็ไม่ได้รับผลกระทบต่อค่าตัวแปร cheese จาก **subtest** เลย ดังนั้นจึงเห็นว่าเมื่อโปรเซสพยายามจะแสดงค่าของตัวแปรซึ่งไม่เคยได้รับการกำหนดค่ามาก่อนเลยดังในกรณีของสคริปต์ **subtest** ค่าของตัวแปรที่ได้รับการแสดงผลออกมาจึงเป็นความว่างเปล่า คราวนี้ผู้เขียนขอเปลี่ยนสคริปต์ **extest1** ไปเป็น **extest2** ซึ่งต่างกันตรงที่สคริปต์ใหม่นี้มีการเพิ่มคำสั่ง export เพื่อส่งค่าตัวแปร cheese ไปยังโปรเซสลูก รายละเอียดและผลลัพธ์ที่ได้จากการเรียกใช้สคริปต์แสดงในกรอบตัวอย่างที่ 9

กรอบที่ 9

```
$ cat extest2
export cheese
cheese=american
echo "extest2 1: $cheese"
subtest
echo "extest2 2: $cheese"
$ extest1
extest2 1: american
subtest 1: american
subtest 2: swiss
extest2 2: american
```

จะเห็นว่าโปรเซสลูก (subtest) ได้รับการส่งผ่านค่าของตัวแปร cheese มาจากสคริปต์ **extest2** หลังจากแสดงค่าของตัวแปรไปแล้วก็จะเปลี่ยนค่าของตัวแปรจาก american ไปเป็น swiss และเมื่อโปรเซสดังกล่าวทำงานเสร็จสิ้นลงแล้วก็จะส่งผ่านการทำงานกลับไปยัง **extest2** ซึ่งถือเป็นโปรเซสแม่และยังรักษาค่าของตัวแปร cheese เป็น american อยู่เช่นเดิม

คำสั่ง **read** สำหรับผู้ที่เริ่มต้นฝึกเขียนเชลล์สคริปต์ไปได้ระยะหนึ่งจะพบว่ามีหลายครั้งที่ตนต้องการให้โปรแกรมของตนสามารถรับข้อความผ่านทางคอนโซลหรือเทอร์มินัลใช้งานของตนได้ โดยข้อความที่ได้รับจะถูกนำไปเก็บไว้ในตัวแปรซึ่งโดยส่วนใหญ่ก็คงจะต้องเป็นตัวแปรชนิดผู้ใช้งานกำหนดเอง ในการนี้จำเป็นต้องมีการใช้คำสั่งเพื่อให้เชลล์ทำการหยุดรอรับการป้อนข้อมูลจากคอนโซลใช้งานแล้วนำข้อมูลที่ได้รับนั้นไปเก็บไว้ในตัวแปร สำหรับระบบปฏิบัติการ UNIX ก็ได้มีการเตรียมคำสั่ง **read** ซึ่งทำหน้าที่ดังกล่าวไว้เป็นการอำนวยความสะดวก ก่อนจะกล่าวถึงรายละเอียดการใช้งาน ขอให้ลองศึกษาการใช้งานคำสั่ง **read** ดังแสดงในกรอบตัวอย่างที่ 10

กรอบที่ 10

```
$ cat read1
echo "Welcome, please enter: \c"
read firstline
echo "You entered: $firstline"
$ read1
Welcome, please enter: This is a test message.
You entered: This is a test message.
```

จากตัวอย่างในบรรทัดแรกเป็นการใช้คำสั่ง **echo** เพื่อพิมพ์ข้อความแจ้งให้ผู้ใช้งานทำการป้อนข้อความ เครื่องหมาย \c ใช้เพื่อป้องกันมิให้คำสั่ง **echo** ทำการขึ้นบรรทัดใหม่ ทั้งนี้เพื่อรอให้มีการป้อนข้อความจากผู้ใช้งาน บรรทัดที่สองเป็นการสั่งให้เชลล์ทำการอ่านข้อความที่ได้รับการป้อนจากผู้ใช้งานไปเก็บไว้ในตัวแปรชื่อ firstline และบรรทัดที่สามจะเป็นการนำค่าตัวแปร firstline มาแสดงออกบนจอภาพ สังเกตว่ารูปแบบในการกำหนดแสดงผลในบรรทัดที่เป็นการเขียนแบบโหวดทั้งนี้เพราะเราไม่อาจทราบได้ว่าผู้ใช้งานจะทำการพิมพ์ข้อความใด ๆ ผ่านเข้ามายังตัวแปร firstline ลองคิดดูว่าหากมีการป้อนข้อความเป็น "*" ผลที่แสดงออกมาบนจอภาพตามคำสั่งในบรรทัดที่สามจะเป็นอย่างไร ลองดูตัวอย่างในกรอบที่ 11 ซึ่งเป็นการแสดงผลลัพธ์ที่ได้จากการป้อนข้อความ * ผ่านเข้าไปในสคริปต์โดยสคริปต์ **read1_no_quote** มีเนื้อความเหมือนกับ **read1** ทุกประการ ต่างกันเพียงบรรทัดที่สามมีเนื้อหาเป็น echo You entered: $firstline

กรอบที่ 11

```
$ read1_no_quote
Go ahead: *
You entered: read1 read1_no_quote script.1
$ ls
read1   read1_no_quote  script.1
```

ผลที่ได้จากการป้อนข้อความ "*" จะทำให้สคริปต์แสดงรายชื่อของไฟล์ทั้งหมดที่มีอยู่ภายใต้ไดเรคทอรีใช้งานปัจจุบันแทนที่จะพิมพ์เครื่องหมาย "*" กลับออกมา ดังนั้นในการเขียนโปรแกรมเชลล์สคริปต์จึงต้องระมัดระวังการเกิดปัญหาดังกล่าวให้ดี สำหรับสคริปต์ **read2** ซึ่งแสดงในกรอบตัวอย่างที่ 12 จะทำการรอรับข้อความ (ซึ่งจะต้องอยู่ในรูปของคำสั่ง) จากผู้ใช้งานไปเก็บไว้ในตัวแปรชื่อ command จากนั้นจึงทำการเรียกใช้งานคำสั่งที่เก็บไว้ในตัวแปร command จากนั้นจึงพิมพ์ข้อความในบรรทัดสุดท้ายแล้วสิ้นสุดการทำงาน ตัวอย่างการทำงานของสคริปต์ **read2** แสดงในกรอบตัวอย่างที่ 13 โดยในตัวอย่างแรกผู้ใช้งานทำการป้อนข้อความเป็นคำสั่ง **echo** ตามด้วยข้อความชุดหนึ่ง และตัวอย่างที่สองเป็นการป้อนข้อความเป็นคำสั่ง **who** เพื่อให้แสดงรายชื่อของที่ทำการ login ในขณะนั้น

กรอบที่ 12

```
$ cat read2
echo "Enter a command: \c"
read command
$command
echo Thanks
```

กรอบที่ 13

```
$ read2
Enter a command: echo Please display this message.
Please display this message.
Thanks
$ read2
Enter a command: who
pairoj          tty01           Sep 27  19:50
wattana         tty06           Sep 27  20:09
Thanks
```

คราวนี้ลองมาประยุกต์ใช้งานคำสั่ง **read** กันบ้าง โดยในครั้งนี้ต้องการสร้างตัวแปรรับข้อความตามหลังคำสั่ง **read** มากกว่า 1 ตัว จากตัวอย่างในกรอบที่ 14 เป็นการแสดงเนื้อหาภายในสคริปต์ **read3** พร้อมทั้งผลลัพธ์ที่ได้จากการเรียกใช้งาน จะเห็นว่ามีการแยกคำแต่ละคำที่ได้รับการป้อนผ่านคอนโซลไปเก็บไว้ในตัวแปร word1, word2 และ word3 ในกรณีที่มีการป้อนข้อความจากคอนโซลเข้ามามากกว่า 3 คำ คำสั่ง **read** จะนำคำแรกและคำที่สองมาเก็บไว้ในตัวแปร word1 และ word2 ตามลำดับ สำหรับกลุ่มคำที่เหลือจะถูกนำรวมไปเก็บไว้ในตัวแปร word3

กรอบที่ 14

```
$ cat read3
echo "Enter something: \c"
read word1 word2 word3
echo "Word 1 is: " $word1"
echo "Word 2 is: " $word2"
echo "Word 3 is: " $word3"
$ read3
Enter something: this is something
Word 1 is: this
Word 2 is: is
Word 3 is: something
$ read3
Enter something: this is something else, really.
Word 1 is: this
Word 2 is: is
Word 3 is: something else, really.
```

อีกสิ่งหนึ่งที่มักพบเห็นใช้งานกันมากในกรณีของเชลล์ชนิดผู้ใช้งานสร้างเองก็คือการใช้ตัวแปรในการเก็บผลลัพธ์ที่ได้จากการใช้คำสั่งมาตรฐานของ UNIX โดยที่เมื่อใดก็ตามที่มีการครอบคำสั่งด้วยเครื่องหมาย ` เชลล์จะทำการแทนที่คำสั่งดังกล่าวพร้อมด้วยเครื่องหมาย

ด้วยค่าผลลัพธ์ที่ได้จากการสั่งใช้งานคำสั่งนั้น เทคนิคดังกล่าวมีชื่อเรียกว่าการแทนที่คำสั่ง (Command Substitution) สำหรับตัวอย่างการประยุกต์ใช้งานเทคนิคดังกล่าวร่วมกับตัวแปรชนิดผู้ใช้งานสร้างเองมีแสดงอยู่ในกรอบที่ 15

กรอบที่ 15

```
$ cat dir
dir=`pwd`
echo "You are using the $dir directory."
$ cat dir
You are using the /export/home/pairoj directory.
```

ทั้งหมดนี้เป็นเรื่องของตัวแปรชนิดผู้ใช้งานสร้างเอง สำหรับตัวแปรประเภทต่อไปที่จะขอกล่าวถึงก็คือตัวแปรมาตรฐานที่ได้รับการสร้างและกำหนดค่าขึ้นโดยเชลล์ในระบบปฏิบัติการ UNIX เอง

ตัวแปรมาตรฐาน

ตัวแปรมาตรฐานส่วนใหญ่จะได้รับการสร้างขึ้นจากกระบวนการทำงานของเชลล์ในช่วงที่มีการ login เข้าสู่ระบบปฏิบัติการ UNIX ผู้ใช้งานสามารถทำการกำหนดค่าใหม่ให้กับตัวแปรมาตรฐานได้ทั้งจากการใช้คำสั่งโดยตรงหรือการแก้ไขค่าของตัวแปรในไฟล์ .profile ซึ่งอยู่ในไดเรคทอรีหลักของผู้ใช้งานแต่ละราย โดยทั่วไปผู้ใช้งานมักต้องการส่งค่าของตัวแปรเหล่านี้ไปยังเชลล์หรือซับเชลล์อื่น ๆ ที่ตนต้องการจะสร้างขึ้น ระบบปฏิบัติการ UNIX จึงได้รับการออกแบบให้ผู้ใช้งานสามารถทำการส่งผ่านค่าของตัวแปรเหล่านี้ไปยังโปรเซสหรือเชลล์หรือซับเชลล์อื่น ๆ ที่จะเกิดตามขึ้นมาได้ ในที่นี้ผู้เขียนจะขอกล่าวถึงตัวแปรมาตรฐานที่มักพบเห็นและใช้งานกันบ่อย ๆ ดังนี้

ตัวแปร HOME

โดยทั่วไปเมื่อผู้ใช้งานทำการ login เข้าสู่ระบบ UNIX ก็จะได้รับการกำหนดไดเรคทอรีเริ่มต้นเพื่อใช้ในการสร้างไฟล์ข้อมูลเป็นของตนเอง เรียกกันว่าไดเรคทอรีผู้ใช้งาน (Home directory) ผู้ที่ทำการกำหนดจัดสรรไดเรคทอรีดังกล่าวก็คือผู้ดูแลระบบโดยรายละเอียดต่าง ๆ จะถูกเก็บอยู่ในไฟล์ /etc/passwd ดังนั้นเมื่อผู้ใช้งานแต่ละรายทำการ login เข้าสู่ระบบปฏิบัติการ เชลล์ที่ทำหน้าที่รองรับการใช้งานในขณะนั้นก็จะทำการอ่านค่าไดเรคทอรีผู้ใช้งานจากไฟล์ /etc/passwd แล้วส่งค่าดังกล่าวไปเก็บไว้ในตัวแปรมาตรฐาน HOME ไม่ว่าหลังจากนั้นผู้ใช้งานจะทำการย้ายไดเรคทอรีใช้งานไปยังตำแหน่งอื่นใดก็ตาม หากมีการพิมพ์คำสั่ง cd โดยไม่มีชื่อไดเรคทอรีต่อท้าย เชลล์จะทำการเติมค่าไดเรคทอรีที่ว่างอยู่ด้วยค่าตัวแปร HOME ทันที ลองดูตัวอย่างในกรอบที่ 16 ครับ

กรอบที่ 16

```
$ pwd
/home/pairoj/document
$ echo $HOME
/home/pairoj
$ echo $HOME
$ pwd
/home/pairoj
```

ตัวแปร PATH

ในการใช้งานระบบปฏิบัติการไม่ว่าจะเป็น DOS หรือ UNIX ก็ตาม บ่อยครั้งที่ผู้ใช้งานต้องการเรียกใช้งานคำสั่งหรือโปรแกรมซึ่งอาจจะอยู่ในไดเรคทอรีใช้งานปัจจุบัน ซึ่งในกรณีนี้ผู้ใช้งานสามารถเรียกชื่อโปรแกรมได้โดยตรงเช่น **startServer** ในกรณีที่ผู้ใช้งานทราบว่าโปรแกรมดังกล่าวอยู่ในไดเรคทอรีอื่นซึ่งมิใช่ไดเรคทอรีใช้งานปัจจุบัน ก็ยังคงสามารถเรียกใช้งานโปรแกรมได้โดยใช้การอ้างถึงแบบสัมบูรณ์ ซึ่งต้องอ้างถึงพาธ (path) ทั้งหมดที่เก็บไฟล์ดังกล่าว เช่น **/opt/appl/runScript/startServer** อย่างไรก็ตาม การอ้างชื่อไฟล์แบบสัมบูรณ์ในลักษณะดังกล่าวค่อนข้างจะไม่สะดวกเนื่องจากต้องพิมพ์คำสั่งยาว ระบบปฏิบัติการ UNIX ได้ออกแบบให้มีการกำหนดตัวแปรมาตรฐานชื่อ PATH ซึ่งใช้ในการเก็บตัวอักษรซึ่งแทนชื่อของกลุ่มพาธที่ต้องการให้อยู่ในรายการค้นหา (search path) โดยเมื่อผู้ใช้งานแต่ละรายทำการ login เข้าสู่ระบบก็จะได้รับการกำหนดตัวแปรมาตรฐาน PATH ไว้ให้ทันที เมื่อผู้ใช้งานพิมพ์เรียกใช้งานคำสั่งหรือโปรแกรมใดที่ไม่ปรากฏว่ามีอยู่ในไดเรคทอรีใช้งานปัจจุบัน เชลล์ก็จะทำการค้นหาไฟล์ดังกล่าวตามพาธที่ระบุไว้ในตัวแปร PATH ไล่ไปจนกว่าจะพบหรือจนกว่าจะครบทุกพาธ หากเชลล์ทำการค้นหาตามพาธทั้งหมดแล้วแต่ไม่พบไฟล์ที่อ้างถึงก็จะทำการแจ้งต่อผู้ใช้งานว่าไม่พบไฟล์ดังกล่าว รายการพาธที่แสดงไว้ในตัวแปร PATH สามารถแก้ไขเปลี่ยนแปลงได้โดยตัวผู้ใช้งานเอง สำหรับระบบปฏิบัติการ SunOS ของบริษัท Sun MicroSystem จะทำการกำหนดค่าตัวแปรเริ่มต้นของตัวแปร PATH ไว้ให้รวมพาธชื่อ /usr/ucb /usr/bin และ /bin ซึ่งผู้ใช้งานสามารถเพิ่มเติมชื่อพาธอื่น ๆ ได้ตามต้องการ

วิธีการเขียนค่าตัวแปรพาธทำได้ไม่ยาก โดยผู้ใช้งานเพียงแต่พิมพ์ชื่อพาธที่ต้องการให้อยู่ในรายการค้นหา แต่ละพาธจะต้องคั่นจากกันด้วยเครื่องหมาย : ลำดับในการค้นหาไฟล์จะเป็นไปตามลำดับชื่อพาธที่เขียนเรียงกันไว้ จากตัวอย่างในกรณีของระบบปฏิบัติการ SunOS การค้นหาจะเริ่มจากภายในพาธชื่อ /usr/ucb หากไม่พบไฟล์ที่ต้องการ เชลล์ก็จะทำการตรวจค้นจากพาธ /usr/bin และ /bin ไปตามลำดับ สำหรับการสร้างหรือแก้ไขเพิ่มเติมรายการพาธในตัวแปร PATH ของเชลล์แบบบอร์นกระทำได้ง่าย ๆ ดังนี้

กรอบที่ 17

```
$ PATH=/usr/ucb:/bin/usr:/home/pairoj
$ export PATH
```

ตัวแปร MAILPATH และ MAILCHECK

เป็นตัวแปรมาตรฐานที่ใช้บอกถึงไดเรคทอรีซึ่งใช้ในการเก็บไฟล์จดหมายอิเล็กทรอนิกส์ โดยทั่วไปไดเรคทอรีดังกล่าวสำหรับผู้ใช้แต่ละรายมักจะเป็น **/var/mail/ชื่อผู้ใช้งาน** หรือ **/var/spool/mail/ชื่อผู้ใช้งาน** ในกรณีของระบบปฏิบัติการ UNIX รุ่นเก่า ๆ มักจะเก็บไฟล์ดังกล่าวไว้ในไดเรคทอรี /usr/mail หรือ /usr/spool/mail สำหรับตัวแปร MAILPATH จะเก็บรายชื่อของไฟล์เขียนแยกจากกันด้วยเครื่องหมาย : ซึ่งผู้ใช้จะกำหนดค่าให้กับตัวแปรนี้หรือไม่ก็ได้ หากมีกำหนดค่าตัวแปรดังกล่าวนี้ไว้เมื่อมีการเปลี่ยนแปลงแก้ไขไฟล์ที่มีการระบุชื่อไว้ เชลล์ก็จะทำการแจ้งเตือนผู้ใช้งานทันที ประโยชน์ที่เห็นได้ชัดก็คือหากมีการระบุชื่อของไฟล์ที่เก็บจดหมายเข้า เมื่อมีจดหมายใหม่เข้ามา ไฟล์ก็จะใหญ่ขึ้น ถือเป็นการเปลี่ยนแปลงขนาดไฟล์ ผู้ใช้ก็จะได้รับการแจ้งเตือนโดยทันที สำหรับตัวแปร MAILCHECK จะใช้ในการกำหนดช่วงความถี่ให้เชลล์ทำการตรวจสอบจดหมายใหม่ ค่ามาตรฐานคือ 600 วินาที

ตัวแปร PS1

ตัวแปร PS1 (Prompt String 1) ใช้แจ้งให้ผู้ใช้งานทราบว่าระบบปฏิบัติการกำลังรอรับคำสั่งอยู่ โดยทั่วไปในกรณีของเชลล์แบบบอร์น การหยุดรอรับคำสั่งของเชลล์จะถูกแทนด้วยการแสดงเครื่องหมายพรอมต์ $ บนหน้าจอ ในกรณีที่ผู้ใช้งานต้องการเปลี่ยนแปลงเครื่องหมายพรอมต์เป็นเครื่องหมายหรือข้อความอื่น ๆ ก็สามารถทำได้โดยการเปลี่ยนแปลงค่าของตัวแปร PS1 ตัวอย่างเช่น ในกรณีที่ผู้ใช้งานทำงานอยู่กับเครื่องคอมพิวเตอร์หลายเครื่องที่เชื่อมต่อกันเป็นระบบเครือข่าย ก็นับเป็นการดีที่จะเปลี่ยนเครื่องหมายพรอมต์ให้เป็นข้อความที่แสดงถึงชื่อของเครื่องที่ตนใช้งานอยู่ ลองดูตัวอย่างในกรอบที่ 18 ซึ่งแสดงให้เห็นถึงวิธีการดังกล่าวในกรณีของเชลล์แบบบอร์น

กรอบที่ 18

```
$ PS1="`hostname `:"
bravo: echo test prompt
test prompt
bravo:
```

ตัวแปร PS2

สำหรับตัวแปร PS2 (Prompt String 2) นั้นสามารถอธิบายได้ไม่ยากนัก โดยขอให้ลองดูตัวอย่างในกรอบที่ 19 จะเห็นว่าคำสั่งในบรรทัดแรกเป็นการใช้ **echo** เพื่อพิมพ์ข้อความให้ปรากฏบนหน้าจอ แต่ผู้เขียนพิมพ์ข้อความไม่จบโดยไม่ทำการปิดเครื่องหมายคำพูดแต่ได้ทำการพิมพ์ ENTER เพื่อขึ้นบรรทัดใหม่ เชลล์จะทราบโดยอัตโนมัติว่ายังต้องรอข้อความในส่วนที่เหลือจึงแสดงเครื่องหมายพรอมต์ที่สอง (secondary prompt) ซึ่งใช้ในกรณีของการรอรับข้อความที่ยังเหลือ โดยทั่วไปจะใช้เครื่องหมายพรอมต์เป็น > เชลล์จะยอมรับข้อความทั้งหมดก็ต่อเมื่อผู้ใช้งานได้ทำการปิดเครื่องหมายคำพูดไปแล้ว ในกรณีที่ต้องการเปลี่ยนเครื่องหมายพรอมต์ก็สามารถทำได้โดยแก้ไขตัวแปร PS2 ดังแสดงในตัวอย่าง

กรอบที่ 19

```
$ echo "Demonstration of prompt string
> 2"
Demonstration of prompt string
2
$ PS2="secondary prompt: "
$ echo "Demonstration of prompt string
secondary prompt : 2"
Demonstration of prompt string
2
$
```

ผ่านไปอีกหนึ่งตอนกับเรื่องราวของตัวแปรชนิดต่าง ๆ ที่มีการใช้งานในระบบปฏิบัติการ UNIX ฉบับหน้าผู้เขียนจะกล่าวถึงตัวแปรเชลล์ที่อ่านค่าได้เพียงอย่างเดียว และเริ่มพูดถึงคำสั่งมาตรฐานที่ใช้การกำหนดเงื่อนไข การทำงานซึ่งถือเป็นเครื่องมือสำคัญต่อการทำ Shell Programming ในระบบปฏิบัติการ UNIX เป็นอย่างมาก แล้วพบกันครับ